1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ**

**по лабораторной работе № 2**

# Cервисы MS Windows

1. по дисциплине «Безопасность современных информационных технологий»
2. Выполнил
3. студент гр. 33508/3 Проценко Е.Г.

\_\_\_\_\_\_\_\_\_\_

Руководитель Иванов Д.В.

\_\_\_\_\_\_\_\_\_\_

1. Санкт-Петербург
2. 2016

1. Содержание

[1 Цель работы 3](#_Toc465194253)

[2 Теоретические сведения 4](#_Toc465194254)

[2.1 Об SCM 4](#_Toc465194255)

[2.2 Сервисные программы 5](#_Toc465194256)

[2.3 Service Entry Point 6](#_Toc465194257)

[2.4 Service ServiceMain Function 7](#_Toc465194258)

[2.5 Service Control Handler 8](#_Toc465194259)

[2.6 SCP 9](#_Toc465194260)

[3 Вывод 10](#_Toc465194261)

[4 Ответы на контрольные вопросы 11](#_Toc465194262)

[4.1 Какие операции выполняются при установке сервиса в систему? 11](#_Toc465194263)

[4.2 Что такое SCP и зачем необходимо его реализовывать? 11](#_Toc465194264)

[4.3 Какие возможности по управлению сервисами НЕ предоставляет оснастка “Службы”? 11](#_Toc465194265)

[Список используемых источников 12](#_Toc465194266)

[Приложение А 13](#_Toc465194267)

# Цель работы

Написать программу-сервер и программу-клиент, работающие под Windows XP и 7. Сервер должен работать в качестве сервиса Windows и предоставлять доступ локальным и удаленным клиентам к файловой системе (локальной и удаленной).

Требования:

* необходимо использовать программу-клиент из лабораторной работы №1 (без изменений ее кода);
* в качестве основы программы-сервер необходимо использовать исходный код программы-сервер из лабораторной работы №1 (statefull сервер);
* работа в Windows XP/7 (все SP);
* инсталляция, деинсталляция, запуск, останов из командной строки (реализация SCP);
* корректный останов и перезапуск службы по запросу от SCM (через оснастку "Службы") (в том числе корректное завершение текущих активных соединений с удаленными клиентами).

# Теоретические сведения

## Об SCM

SCM (Service Control Manager) запускается вместе с системой. Он является RPC сервером, так SCM может управлять сервисами и контролирующими программами на удаленных компьютерах.

SCM предоставляет интерфейс для следующих задач:

* содержание базы данных установленных сервисов;
* запуск сервисов при загрузке системы или по требованию;
* содержание состояния работающих сервисов;
* передача управляющих запросов сервисам;
* блокировка и разблокировка базы данных сервисов.

SCM поддерживает типа handle, который предоставляет доступ к следующим объектам:

* база данных установленных сервисов;
* сами сервисы;
* замок базы данных.

SCManager объект является объектом базы данных. Та, в свою очередь, является контейнером для объектов сервисов. Функция **OpenSCManager** возвращает handle на объект SCManager на определенном компьютере. Этот handle используется для установки, удаления, открытия и перечисления сервисов, а также для работы с замком.

Объект сервиса определяет установленный сервис. Функции **CreateService** и **OpenService** возвращают такой handle.

**OpenSCManager, CreateService, OpenService** могут запросить разный уровень доступа к SCManager и сервисному объектам. Запрошенный доступ разрешается или запрещается, в зависимости от маркера доступа вызывающего процесса, и дескриптора безопасности соответствующего SCManager или сервисного объекта.

**CloseServiceHandle** закрывает handles от SCManager и сервисного объекта.

SCM содержит базу данных установленных сервисов и driver services и предоставляет единый и безопасный метод управления ими. База данных содержит информацию о том, как каждый сервис или сервис драйвера должен быть запущен.

## Сервисные программы

Сервисные программы содержат исполняемый код для одного или более сервисов. Сервисная программа, созданная с типом SERVICE\_WIN32\_OWN\_PROCESS содержит код только одного сервиса.Сервисная программа, созданная с типом SERVICE\_WIN32\_SHARE\_PROCESS содержит код для более чем одного сервиса, позволяющий им делить код. Пример сервисной программы, которая занимается этим – это generic service host process, Svchost.exe, which hosts internal Windows services. Заметьте, что Svchost.exe зарезервирован для использования операционной системой и не может использоваться не Windows сервисами. Вместо этого, разработчики должны создавать их собственные сервисные хостовые программы.

Сервисные программы могут быть сконфигурированы так, чтобы работать в контексте пользовательского аккаунта локального, primary или trusted домена. Так же оно может быть сконфигурировано так, чтобы запускаться в специальном сервисном пользовательском режиме.

Следующие топики описывают требуемый интерфейс SCM, который сервисная программа должна иметь:

* [Service Entry Point](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685477(v=vs.85).aspx)
* [Service ServiceMain Function](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685984(v=vs.85).aspx)
* [Service Control Handler Function](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685149(v=vs.85).aspx)

## Service Entry Point

Сервисы в основном написаны как консольное приложения. Entry point консольного приложения это функция **main**. **Main** получает аргументы from **ImagePath** value from the registry key for the service. Больше информации здесь:  [CreateService](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682450(v=vs.85).aspx) function.

КогдаSCM запускает сервисную программу, она ожидает когда та вызовет функцию **StartServiceCtrlDispatcher**. Рекомендации:

* Сервис типа SERVICE\_WIN32\_OWN\_PROCESS должен вызывать StartServiceCtrlDispatcher сразу, из главного потока. Вы можете выполнять любые инициализации после того, как сервер начнет работу, как описано в [Service ServiceMain Function](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685984(v=vs.85).aspx).
* Если тип сервиса SERVICE\_WIN32\_SHARE\_PROCESS и нужна общая инициализации для всех сервисов в программе, вы можете выполнить инициализацию в main thread перед вызовом **StartServiceCtrlDispatcher**, если это занимает менее 30 секунд. Однако, вы должны создать другой поток, чтобы сделать общую инициализацию, пока главный поток вызывает StartServiceCtrlDispatcher**.** Но по-прежнему все специфичные настройки должны быть выполнены после начала работы сервиса.

Функция **StartServiceCtrlDispatcher** принимает на вход структуру  [**SERVICE\_TABLE\_ENTRY**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms686001(v=vs.85).aspx) для каждого сервиса, содержащегося в программе. Каждая структура определяет имя сервиса и entry point для него. For an example, see [Writing a Service Program's main Function](https://msdn.microsoft.com/en-us/library/windows/desktop/ms687416(v=vs.85).aspx).

Если функция **StartServiceCtrlDispatcher** успешно выполнилась, то вызывающий поток не возобновит работу, пока для всех работающих сервисов в процессе не будет получено состояние **SERVICE\_STOPPED.** SCM отправляет контролирующие запросы к этому потоку через именованый pipe. Поток выступает в качестве диспетчера управления, выполняя следующие задачи:

* Создание нового потока для вызова соответствующего entry point, когда новый сервис стартует.
* Вызов соответствующей [handler function](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685149(v=vs.85).aspx), чтобы обрабатывать сервисные контролирующие запросы.

## Service ServiceMain Function

Когда сервисная контролирующая программа посылает запрос о том, что новый сервис запустился, SCM запускает его и отправляет запрос на запуск на control dispatcher. Control dispatcher создает новый поток, чтобы запустить там [ServiceMain](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685138(v=vs.85).aspx) function для сервиса. For an example, see [Writing a ServiceMain Function](https://msdn.microsoft.com/en-us/library/windows/desktop/ms687414(v=vs.85).aspx).

The [ServiceMain](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685138(v=vs.85).aspx) function должна выполнять следующие задачи:

1. Инициализация глобальных переменных
2. Вызов функции  [RegisterServiceCtrlHandler](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685054(v=vs.85).aspx) немедленно, чтобы зарегистрировать [Handler](https://msdn.microsoft.com/en-us/library/windows/desktop/ms683240(v=vs.85).aspx) function, чтобы обрабатывать контролирующие запросы для сервиса. Возвращаемое значение **RegisterServiceCtrlHandler** - service status handle, который будет использоваться в тех вызовах, которые оповещают SCM о статусе сервиса.
3. Выполнять инициализацию. Если время исполнения инициализации ожидается очень коротким (менее одной секунды), то инициализация может быть выполнена прямо в **ServiceMain.** Если время инициализации ожидается дольше, чем одна секнда, то сервис должен использовать одну из следующих техник инициализации:

* Вызываем функцию  [**SetServiceStatus**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms686241(v=vs.85).aspx), чтобы сообщить, что SERVICE\_RUNNING, но не принимает никакие controls, пока не закончена инициализация. The service does this by calling **SetServiceStatus** with **dwCurrentState** set to SERVICE\_RUNNING and **dwControlsAccepted** set to 0 in the [**SERVICE\_STATUS**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685996(v=vs.85).aspx) structure. Это гарантирует, что SCM не отправит никакие контролирующие запросы сервису, пока тот не готов и освобождает SCM для того, чтобы он занимался другими сервисами. Такой подход рекомендован для повышения производительности, особенно для autostart смервисов.

1. Когда инициализация завершена, вызовите **SetServiceStatus,** чтобы задать SERVICE\_RUNNING статус сервиса и определить controls, которые сервис готов принимать. For a list of controls, see the [**SERVICE\_STATUS**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685996(v=vs.85).aspx) structure.
2. Выполните задачи обслуживания или, если нет тасков в ожидании, верните контроль вызывающему. Любые изменения в состоянии сервиса являются основаниями для вызова **SetServiceStatus,** чтобы сообщить информацию о новом состоянии.
3. Если произошла ошибка вовремя инициализации или работы сервиса, сервис должен вызвать **SetServiceStatus**, чтобы обозначить состояние сервиса как SERVICE\_STOP\_PENDING, если очистка будет долгой. После того, как чистка окончена, вызовите **SetServiceStatus,** чтобы обозначит состояние сервиса как SERVICE\_STOPPED from the last thread to terminate. Be sure to set the **dwServiceSpecificExitCode** and **dwWin32ExitCode** members of the [**SERVICE\_STATUS**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685996(v=vs.85).aspx) structure to identify the error.

## Service Control Handler

Каждый сервис имеет control handler, the [**Handler**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms683240(v=vs.85).aspx) function, которая вызывается by control dispatcher, когда сервисный процесс получает control request от service control program. Therefore, this function executes in the context of the control dispatcher. For an example, see [Writing a Control Handler Function](https://msdn.microsoft.com/en-us/library/windows/desktop/ms687413(v=vs.85).aspx).

A service calls the [**RegisterServiceCtrlHandler**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685054(v=vs.85).aspx) or [**RegisterServiceCtrlHandlerEx**](https://msdn.microsoft.com/en-us/library/windows/desktop/ms685058(v=vs.85).aspx)function to register its service control handler function.

Когда control handler вызван, сервис должен вызвать SetServiceStatus, чтобы сообщить его стату для SCM только если обработка контролирующего кода приводит к сменен статуса сервиса. Вызывать **SetServiceStatus**  не обязательно.

## SCP

Для установки, удаления, запуска и останова сервисов нужно работать с SCM, чтобы начать работу с ним нужно выполнить функцию **OpenSCManager,** по результатам которой мы получим handle на SCM.

Оттуда мы можем достать интересующий нас сервис функцией **OpenService.** Так мы получим handle на сервис.

Теперь, чтобы удалить, запустить, остановить сервис нужно вызвать **DeleteService, StartService, ControlService,** передав аргументов handle на сервис.

Чтобы создать сервис понадобится функция **CreateService.**

# Вывод

Я познакомился с реализацией сервисов и сервисных приложений.

Так же в интернете были найдены исходные коды для dns, dhcp, ftp, tftp и других сервисов.

В основном примеры кода пришлось искать на github’е, а также смотреть спецификации на MSDN.

# Ответы на контрольные вопросы

## Какие операции выполняются при установке сервиса в систему?

Функция **CreateService** создает сервисный объект и устанавливает его в SCM, занося в базу данных, создавая ключ в реестре с соответствующим именем, как и будущий сервис, по следующему пути:

HKEY\_LOCAL\_MACHINE\System\CurrentControlSer\Services

Также в базе данных сохраняется конфигурационная и дополнительная информация, такая как: зависимости, описание, группы, путь к образу, тип загрузки, отображаемое имя, тип сервиса.

## Что такое SCP и зачем необходимо его реализовывать?

SCP (Service Control Programs) – запускают и контролируют сервисы. SCP предоставляют следующие возможности: запуск сервиса, отправление контролирующих запросов на сервис, запрос текущего состояния сервиса.

## Какие возможности по управлению сервисами НЕ предоставляет оснастка “Службы”?

Оснастка “Службы” не предоставляет возможности установки и удаления сервиса.

О программе, управляющей сервисом предоставляют только путь к ее образу, больше никакой дополнительной информации о самом процессе. Ее можно посмотреть в специальных программах типа Process Explorer или Process Hacker.

# Список используемых источников

1. <http://www.codeproject.com/Articles/499465/Simple-Windows-Service-in-Cplusplus>
2. [http://stackoverflow.com/](https://www.sans.org/security-resources/policies/general/pdf/acceptable-encryption-policy)
3. <https://github.com/>
4. <https://github.com/sunnyden/reactos/blob/81dce7ad39f0fa5584d0cf1727443d77835e7d7e/base/services/tftpd/tftpd.cpp> - исходные коды сервиса tftp-сервера
5. [https://msdn.microsoft.com/](https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa376391(v=vs.85).aspx)

# Приложение А

#include <iostream>

#include "Example1.h"

#include <Windows.h>

#include <stdio.h>

#include <windows.h>

#include <tchar.h>

#include "accctrl.h"

#include "aclapi.h"

**char** \* **get\_owner**(**const** **char** \* file\_name);

**bool** login\_status = false;

**char** login\_actual[**256**] = "";

**char** \* opened\_file[**256**];

**FILE** \*filik = NULL;

**int** download\_size;

**int** **LogIn**(**const** **char** \* login, **const** **char** \* password)

{

std::cout << login << " logged in" <<login\_status << std::endl;

**if** (login\_status == true) **return** -**1**;

BOOL status;

HANDLE token;

status = LogonUser(login, //

".",

password, //

LOGON32\_LOGON\_NETWORK,

LOGON32\_PROVIDER\_DEFAULT,

&token);

**if** (!status)

{

std::cout << "LogonUser error: " << GetLastError() << std::endl;

**return** GetLastError();

}

status = ImpersonateLoggedOnUser(token);

**if** (!status)

{

std::cout << "Impersonation error: " << GetLastError() << std::endl;

**return** GetLastError();

}

login\_status = true;

strcpy(login\_actual,login);

**return** **0**;

}

**int** **LogOut**()

{

BOOL status;

status = RevertToSelf();

**if** (!status)

{

std::cout << "RevertingToSelf error: " << GetLastError() << std::endl;

**return** GetLastError();

}

std::cout << login\_actual << " logged out!" << std::endl;

login\_status = false;

strcpy(login\_actual,"");

**return** **0**;

}

**int** **CreateFileJP**(**const** **char** \* file\_name)

{

std::cout << "Creating file JP" << std::endl;

**if** (login\_status == false) **return** -**1**;

**FILE** \*f;

f = fopen( file\_name, "rb" );

**if** ( f != **0** )

{

std::cout << "file exists" << std::endl;

fclose( f );

**return** -**1**;

}**else**

{

std::cout << "file\_not\_exist" << std::endl;

//fclose( f );

}

filik = fopen(file\_name, "wb");

**if** (!filik) // если есть доступ к файлу,

{

**return** -**1**;

}

//fclose(opened\_file);

**return** **0**;

}

**int** **OpenFileJP**(**const** **char** \* file\_name)

{

**if** (login\_status == false) **return** -**1**;

filik = fopen( file\_name, "rb" );

**if** ( filik != **0** )

{

std::cout << "file exists" << std::endl;

fseek(filik, **0**, SEEK\_END);//перемещает указатель, соответствующий потоку hFile, на новое место расположения отстоящее от SEEK\_END на 0 байтов.

download\_size = ftell(filik);

//if (file\_size == -1) return -4;

fseek(filik, **0**, SEEK\_SET);

}

**else**

{

std::cout << "file\_not\_exist" << std::endl;

fclose( filik );

**return** -**1**;

//fclose( f );

}

//fclose(opened\_file);

**return** **0**;

}

**int** **Upload**(**int** buffer,**int** type)

{

/\*FILE \*out;

out = fopen((char\*)file\_name,"ab");

fwrite(buffer,sizeof(char),size,out);

fclose(out);\*/

//FILE \*hFile;

//fopen\_s(&filik, (char \*)buffer, "ab");

//printf("%x", buffer);

/\*for (int i = 0; i < size; i++)

fprintf(hFile, "%c", buffer[i]);

fclose(hFile); \*/

**if** (type == **2**)

fclose(filik);

**else**

fputc(buffer, filik);

**return** **0**;

}

**char** \* **get\_owner**(**const** **char** \* file\_name)

{

DWORD dwRtnCode = **0**;

PSID pSidOwner = NULL;

BOOL bRtnBool = TRUE;

LPTSTR AcctName = NULL;

LPTSTR DomainName = NULL;

DWORD dwAcctName = **1**, dwDomainName = **1**;

SID\_NAME\_USE eUse = SidTypeUnknown;

HANDLE hFile;

PSECURITY\_DESCRIPTOR pSD = NULL;

// Get the handle of the file object.

hFile = CreateFile(

TEXT(file\_name),

GENERIC\_READ,

FILE\_SHARE\_READ,

NULL,

OPEN\_EXISTING,

SECURITY\_ANONYMOUS,

NULL);

// Check GetLastError for CreateFile error code.

**if** (hFile == INVALID\_HANDLE\_VALUE) {

DWORD dwErrorCode = **0**;

dwErrorCode = GetLastError();

\_tprintf(TEXT("CreateFile error = %d**\n**"), dwErrorCode);

**return** NULL;

}

// Get the owner SID of the file.

dwRtnCode = GetSecurityInfo(

hFile,

SE\_FILE\_OBJECT,

OWNER\_SECURITY\_INFORMATION,

&pSidOwner,

NULL,

NULL,

NULL,

&pSD);

CloseHandle(hFile);

// Check GetLastError for GetSecurityInfo error condition.

**if** (dwRtnCode != ERROR\_SUCCESS) {

DWORD dwErrorCode = **0**;

dwErrorCode = GetLastError();

\_tprintf(TEXT("GetSecurityInfo error = %d**\n**"), dwErrorCode);

**return** NULL;

}

// First call to LookupAccountSid to get the buffer sizes.

bRtnBool = LookupAccountSid(

NULL, // local computer

pSidOwner,

AcctName,

(LPDWORD)&dwAcctName,

DomainName,

(LPDWORD)&dwDomainName,

&eUse);

// Reallocate memory for the buffers.

AcctName = (LPTSTR)GlobalAlloc(

GMEM\_FIXED,

dwAcctName);

// Check GetLastError for GlobalAlloc error condition.

**if** (AcctName == NULL) {

DWORD dwErrorCode = **0**;

dwErrorCode = GetLastError();

\_tprintf(TEXT("GlobalAlloc error = %d**\n**"), dwErrorCode);

**return** NULL;

}

DomainName = (LPTSTR)GlobalAlloc(

GMEM\_FIXED,

dwDomainName);

// Check GetLastError for GlobalAlloc error condition.

**if** (DomainName == NULL) {

DWORD dwErrorCode = **0**;

dwErrorCode = GetLastError();

\_tprintf(TEXT("GlobalAlloc error = %d**\n**"), dwErrorCode);

**return** NULL;

}

// Second call to LookupAccountSid to get the account name.

bRtnBool = LookupAccountSid(

NULL, // name of local or remote computer

pSidOwner, // security identifier

AcctName, // account name buffer

(LPDWORD)&dwAcctName, // size of account name buffer

DomainName, // domain name

(LPDWORD)&dwDomainName, // size of domain name buffer

&eUse); // SID type

// Check GetLastError for LookupAccountSid error condition.

**if** (bRtnBool == FALSE) {

DWORD dwErrorCode = **0**;

dwErrorCode = GetLastError();

**if** (dwErrorCode == ERROR\_NONE\_MAPPED)

\_tprintf(TEXT

("Account owner not found for specified SID.**\n**"));

**else**

\_tprintf(TEXT("Error in LookupAccountSid.**\n**"));

**return** NULL;

} **else** **if** (bRtnBool == TRUE)

// Print the account name.

\_tprintf(TEXT("Account owner = %s**\n**"), AcctName);

**return** AcctName;

}

**int** **Delete**(**const** **char** \* file\_name)

{

**if** (login\_status == false) **return** -**1**;

**char** \* owner = get\_owner(file\_name);

std::cout << "owner: " << owner << std::endl;

std::cout << "login\_actual: " << login\_actual << std::endl;

**if** (strcmp(owner, login\_actual) == **0**)

{

**if**(remove(file\_name)) {

**return** -**2**;

}

**else** **return** **0**;

}

**else** **return** -**3**;

}

**int** **Download**(**int** \* buffer)

{

**if** (download\_size == **0**){

fclose(filik);

filik = NULL;

**return** **1**;

}

download\_size--;

\*buffer = fgetc(filik);

//std::cout << download\_size << " ";

**if** (\*buffer != EOF) **return** **0**;

**else**

{

fclose(filik);

filik = NULL;

**return** **1**;

}

}

// Naive security callback.

RPC\_STATUS CALLBACK **SecurityCallback**(RPC\_IF\_HANDLE /\*hInterface\*/, **void**\* /\*pBindingHandle\*/)

{

**return** RPC\_S\_OK; // Always allow anyone.

}

**int** **rpc**()

{

RPC\_STATUS status;

// Uses the protocol combined with the endpoint for receiving

// remote procedure calls.

status = RpcServerUseProtseqEp(

**reinterpret\_cast**<**unsigned** **char**\*>("ncacn\_ip\_tcp"), // Use TCP/IP protocol.

RPC\_C\_PROTSEQ\_MAX\_REQS\_DEFAULT, // Backlog queue length for TCP/IP.

**reinterpret\_cast**<**unsigned** **char**\*>("4747"), // TCP/IP port to use.

NULL); // No security.

**if** (status)

exit(status);

// Registers the Example1 interface.

status = RpcServerRegisterIf2(

Example1\_v1\_0\_s\_ifspec, // Interface to register.

NULL, // Use the MIDL generated entry-point vector.

NULL, // Use the MIDL generated entry-point vector.

RPC\_IF\_ALLOW\_CALLBACKS\_WITH\_NO\_AUTH, // Forces use of security callback.

RPC\_C\_LISTEN\_MAX\_CALLS\_DEFAULT, // Use default number of concurrent calls.

(**unsigned**)-**1**, // Infinite max size of incoming data blocks.

SecurityCallback); // Naive security callback.

**if** (status)

exit(status);

// Start to listen for remote procedure

// calls for all registered interfaces.

// This call will not return until

// RpcMgmtStopServerListening is called.

status = RpcServerListen(

**1**, // Recommended minimum number of threads.

RPC\_C\_LISTEN\_MAX\_CALLS\_DEFAULT, // Recommended maximum number of threads.

FALSE); // Start listening now.

**if** (status)

exit(status);

}

// Memory allocation function for RPC.

// The runtime uses these two functions for allocating/deallocating

// enough memory to pass the string to the server.

**void**\* \_\_RPC\_USER **midl\_user\_allocate**(**size\_t** size)

{

**return** malloc(size);

}

// Memory deallocation function for RPC.

**void** \_\_RPC\_USER **midl\_user\_free**(**void**\* p)

{

free(p);

}

SERVICE\_STATUS g\_ServiceStatus = {**0**};

SERVICE\_STATUS\_HANDLE g\_StatusHandle = NULL;

HANDLE g\_ServiceStopEvent = INVALID\_HANDLE\_VALUE;

VOID WINAPI **ServiceMain** (DWORD argc, LPTSTR \*argv);

VOID WINAPI **ServiceCtrlHandler** (DWORD);

DWORD WINAPI **ServiceWorkerThread** (LPVOID lpParam);

#define SERVICE\_NAME \_T("BSIT2")

**void** **InstallService**()

{

SC\_HANDLE serviceControlManager = OpenSCManager(**0**, **0**, SC\_MANAGER\_CREATE\_SERVICE);

**if** (serviceControlManager)

{

TCHAR path[\_MAX\_PATH + **1**];

**if** (GetModuleFileName(**0**, path, **sizeof**(path) / **sizeof**(path[**0**])))

{

SC\_HANDLE service = CreateService(serviceControlManager,

SERVICE\_NAME, SERVICE\_NAME,

SERVICE\_ALL\_ACCESS, SERVICE\_WIN32\_OWN\_PROCESS,

SERVICE\_AUTO\_START, SERVICE\_ERROR\_IGNORE, path,

**0**, **0**, **0**, **0**, **0**);

**if** (service)

{

printf("OK: Install service**\n**");

CloseServiceHandle(service);

}

**else**

{

printf("ERROR: Install service. Error code: %d**\n**", GetLastError());

}

}

CloseServiceHandle(serviceControlManager);

}

}

**void** **UninstallService**()

{

SC\_HANDLE serviceControlManager = OpenSCManager(**0**, **0**, SC\_MANAGER\_CONNECT);

**if** (serviceControlManager)

{

SC\_HANDLE service = OpenService(serviceControlManager,

SERVICE\_NAME, SERVICE\_QUERY\_STATUS | DELETE);

**if** (service)

{

SERVICE\_STATUS serviceStatus;

**if** (QueryServiceStatus(service, &serviceStatus))

{

**if** (serviceStatus.dwCurrentState == SERVICE\_STOPPED)

{

**if** (!DeleteService(service))

{

printf("ERROR: Delete service**\n**");

**return**;

}

**else**

{

printf("OK: Delete service**\n**");

}

}

}

CloseServiceHandle(service);

}

CloseServiceHandle(serviceControlManager);

}

}

**void** **RunService**()

{

SERVICE\_STATUS\_PROCESS ssStatus;

SC\_HANDLE serviceControlManager = OpenSCManager(**0**, **0**, SC\_MANAGER\_CONNECT);

DWORD dwBytesNeeded;

**if** (serviceControlManager)

{

SC\_HANDLE hService = OpenService(serviceControlManager, SERVICE\_NAME, SERVICE\_ALL\_ACCESS);

**if** (hService)

{

**if** (!QueryServiceStatusEx(

hService, // handle to service

SC\_STATUS\_PROCESS\_INFO, // information level

(LPBYTE)&ssStatus, // address of structure

**sizeof**(SERVICE\_STATUS\_PROCESS), // size of structure

&dwBytesNeeded)) // size needed if buffer is too small

{

printf("QueryServiceStatusEx failed (%d)**\n**", GetLastError());

CloseServiceHandle(hService);

CloseServiceHandle(serviceControlManager);

**return**;

}

**if** (ssStatus.dwCurrentState != SERVICE\_STOPPED && ssStatus.dwCurrentState != SERVICE\_STOP\_PENDING)

{

printf("Cannot start the service because it is already running**\n**");

CloseServiceHandle(hService);

CloseServiceHandle(serviceControlManager);

**return**;

}

**if** (!StartService(hService, NULL, NULL))

{

printf("ERROR: Start service. Error code: %d**\n**", GetLastError());

**return**;

}

**else**

{

printf("OK: Start service**\n**");

}

}

}

CloseServiceHandle(serviceControlManager);

}

**void** **StopService**()

{

SC\_HANDLE serviceControlManager = OpenSCManager(**0**, **0**, SC\_MANAGER\_CONNECT);

**if** (serviceControlManager)

{

SC\_HANDLE hService = OpenService(serviceControlManager, SERVICE\_NAME, SERVICE\_STOP);

**if** (hService)

{

SERVICE\_STATUS ss;

**if** (!ControlService(hService, SERVICE\_CONTROL\_STOP, &ss))

{

printf("ERROR: Stop service : %d**\n**", GetLastError());

**return**;

}

**else**

{

printf("OK: Stop service**\n**");

}

}

}

CloseServiceHandle(serviceControlManager);

}

**int** **\_tmain** (**int** argc, **char** \*argv[])

{

**if** (argc > **1**)

{

**if** (strcmp(argv[**1**], "-install") == **0**)

InstallService();

**else** **if** (strcmp(argv[**1**], "-uninstall") == **0**)

UninstallService();

**else** **if** (strcmp(argv[**1**], "-run") == **0**)

RunService();

**else** **if** (strcmp(argv[**1**], "-stop") == **0**)

StopService();

}

//else

//{

SERVICE\_TABLE\_ENTRY ServiceTable[] =

{

{SERVICE\_NAME, (LPSERVICE\_MAIN\_FUNCTION) ServiceMain},

{NULL, NULL}

};

**if** (StartServiceCtrlDispatcher (ServiceTable) == FALSE)

{

**return** GetLastError ();

}

**return** **0**;

//}

}

VOID WINAPI **ServiceMain** (DWORD argc, LPTSTR \*argv)

{

DWORD Status = E\_FAIL;

// Register our service control handler with the SCM

g\_StatusHandle = RegisterServiceCtrlHandler (SERVICE\_NAME, ServiceCtrlHandler);

**if** (g\_StatusHandle == NULL)

{

**goto** EXIT;

}

// Tell the service controller we are starting

ZeroMemory (&g\_ServiceStatus, **sizeof** (g\_ServiceStatus));

g\_ServiceStatus.dwServiceType = SERVICE\_WIN32\_OWN\_PROCESS;

g\_ServiceStatus.dwControlsAccepted = **0**;

g\_ServiceStatus.dwCurrentState = SERVICE\_START\_PENDING;

g\_ServiceStatus.dwWin32ExitCode = **0**;

g\_ServiceStatus.dwServiceSpecificExitCode = **0**;

g\_ServiceStatus.dwCheckPoint = **0**;

**if** (SetServiceStatus (g\_StatusHandle , &g\_ServiceStatus) == FALSE)

{

OutputDebugString(\_T(

"My Sample Service: ServiceMain: SetServiceStatus returned error"));

}

/\*

\* Perform tasks necessary to start the service here

\*/

// Create a service stop event to wait on later

g\_ServiceStopEvent = CreateEvent (NULL, FALSE, FALSE, NULL);

**if** (g\_ServiceStopEvent == NULL)

{

// Error creating event

// Tell service controller we are stopped and exit

g\_ServiceStatus.dwControlsAccepted = **0**;

g\_ServiceStatus.dwCurrentState = SERVICE\_STOPPED;

g\_ServiceStatus.dwWin32ExitCode = GetLastError();

g\_ServiceStatus.dwCheckPoint = **1**;

**if** (SetServiceStatus (g\_StatusHandle, &g\_ServiceStatus) == FALSE)

{

OutputDebugString(\_T(

"My Sample Service: ServiceMain: SetServiceStatus returned error"));

}

**goto** EXIT;

}

// Tell the service controller we are started

g\_ServiceStatus.dwControlsAccepted = SERVICE\_ACCEPT\_STOP | SERVICE\_ACCEPT\_PAUSE\_CONTINUE;

g\_ServiceStatus.dwCurrentState = SERVICE\_RUNNING;

g\_ServiceStatus.dwWin32ExitCode = **0**;

g\_ServiceStatus.dwCheckPoint = **0**;

**if** (SetServiceStatus (g\_StatusHandle, &g\_ServiceStatus) == FALSE)

{

OutputDebugString(\_T(

"My Sample Service: ServiceMain: SetServiceStatus returned error"));

}

/\*// Start a thread that will perform the main task of the service

HANDLE hThread = CreateThread (NULL, 0, ServiceWorkerThread, NULL, 0, NULL);

// Wait until our worker thread exits signaling that the service needs to stop

WaitForSingleObject (hThread, INFINITE);\*/

rpc();

/\*

\* Perform any cleanup tasks

\*/

/\*RevertToSelf();

RpcMgmtStopServerListening(NULL);

if (filik)

{

fclose(filik);

filik = NULL;

}

login\_status = false;\*/

CloseHandle (g\_ServiceStopEvent);

// Tell the service controller we are stopped

g\_ServiceStatus.dwControlsAccepted = **0**;

g\_ServiceStatus.dwCurrentState = SERVICE\_STOPPED;

g\_ServiceStatus.dwWin32ExitCode = **0**;

g\_ServiceStatus.dwCheckPoint = **3**;

**if** (SetServiceStatus (g\_StatusHandle, &g\_ServiceStatus) == FALSE)

{

OutputDebugString(\_T(

"My Sample Service: ServiceMain: SetServiceStatus returned error"));

}

**EXIT:**

**return**;

}

// https://github.com/sunnyden/reactos/blob/81dce7ad39f0fa5584d0cf1727443d77835e7d7e/base/services/tftpd/tftpd.cpp

VOID WINAPI **ServiceCtrlHandler** (DWORD CtrlCode)

{

**switch** (CtrlCode)

{

**case** SERVICE\_CONTROL\_INTERROGATE:

**break**;

**case** SERVICE\_CONTROL\_SHUTDOWN:

**case** SERVICE\_CONTROL\_STOP :

**if** (g\_ServiceStatus.dwCurrentState != SERVICE\_RUNNING)

**break**;

/\*

\* Perform tasks necessary to stop the service here

\*/

RevertToSelf();

**if** (filik)

{

fclose(filik);

filik = NULL;

}

login\_status = false;

//RpcMgmtStopServerListening(NULL);

g\_ServiceStatus.dwControlsAccepted = **0**;

g\_ServiceStatus.dwCurrentState = SERVICE\_STOP\_PENDING;

g\_ServiceStatus.dwWin32ExitCode = **0**;

g\_ServiceStatus.dwCheckPoint = **4**;

**if** (SetServiceStatus (g\_StatusHandle, &g\_ServiceStatus) == FALSE)

{

OutputDebugString(\_T(

"My Sample Service: ServiceCtrlHandler: SetServiceStatus returned error"));

}

// This will signal the worker thread to start shutting down

SetEvent (g\_ServiceStopEvent);

**return**;

**case** SERVICE\_CONTROL\_PAUSE:

**break**;

**case** SERVICE\_CONTROL\_CONTINUE:

**break**;

**default:**

**if** (CtrlCode >= **128** && CtrlCode <= **255**)

// user defined control code

**break**;

**else**

// unrecognised control code-

**break**;

}

SetServiceStatus(g\_StatusHandle, &g\_ServiceStatus);

}

DWORD WINAPI **ServiceWorkerThread** (LPVOID lpParam)

{

// Periodically check if the service has been requested to stop

**while** (WaitForSingleObject(g\_ServiceStopEvent, **0**) != WAIT\_OBJECT\_0)

{

/\*

\* Perform main service function here

\*/

// Simulate some work by sleeping

Sleep(**3000**);

}

**return** ERROR\_SUCCESS;

}